

Netzwerke und Flüsse

Kapitel 1: Flussalgorithmen

Hilfreiche Literatur:

- Cormen, Leiserson, Rivest: Introduction to Algorithms, First Edition. MIT Press, 1990.
- Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, Second Edition. MIT Press, 2001.
- Ottmann, Widmayer: Algorithmen und Datenstrukturen. BI-Wiss.-Verl. 1990.

Ein Flussnetzwerk ist ein gerichteter Graph $G = (V, E, c)$ mit zwei ausgewählten Knoten $q, s \in V$ und einer Kapazitätsfunktion $c : E \rightarrow \mathbb{N}_0$. Die Quelle q hat Eingangsgrad 0 und die Senke s hat Ausgangsgrad 0.

Wir definieren $n = |V|$, $m = |E|$ und nehmen an, jeder Knoten ist von q erreichbar. Es gilt $n - 1 \leq m \leq n(n - 1)$.

Ein Fluss in einem Netzwerk G ist eine realwertige Funktion $f : E \rightarrow \mathbb{R}_0^+$ mit den Eigenschaften

a) Flusserhaltung: $\forall u \in V \setminus \{q, s\} :$

$$\underbrace{\sum_{v : (v,u) \in E} f(v,u)}_{f_{\text{in}}(u)} = \underbrace{\sum_{v : (u,v) \in E} f(u,v)}_{f_{\text{out}}(u)} .$$

b) Kapazitätsbeschränkung: $\forall e \in E : f(e) \leq c(e)$.

Der Wert des Flusses f ist definiert als $w(f) = f_{\text{out}}(q)$. Wegen der Flusserhaltung gilt somit auch $w(f) = f_{\text{in}}(s)$.

Problem 1 (Maximaler Fluss) Gegeben sei ein Flussnetzwerk G . Berechne einen maximalen Fluss auf G , d.h. einen Fluss mit größtmöglichem Wert.

Beachte, wir haben angenommen, dass die Kantenkapazitäten ganzzahlig sind. Das ist keine besondere Einschränkung, denn bei rationalen Kapazitäten können wir immer alle Zahlen mit dem Hauptnenner multiplizieren und somit ein Flussproblem mit rationalen Kapazitäten in ein Problem mit ganzzahligen Kapazitäten transformieren.

Die Algorithmen zur Berechnung von maximalen Flüssen, die wir im Folgenden vorstellen werden, haben zahlreiche praktische Anwendungen, die auf den ersten Blick nicht immer unbedingt unmittelbar an Flussprobleme erinnern. In den Übungen präsentieren wir dazu einige, teilweise überraschende Beispiele, wie z.B. das Meisterschaftsproblem.

Die Ford-Fulkerson-Methode

Der folgende algorithmische Rahmen zur Berechnung eines maximalen Flusses f auf einem Flussnetzwerk $G = (V, E, c)$ geht zurück auf Ford und Fulkerson, 1956.

Ford-Fulkerson-Methode:

- 1 $\forall e \in E : f(e) = 0;$
- 2 Solange es einen fv-Weg W gibt
- 3 erhöhe den Fluss f entlang von W maximal;
- 4 Ausgabe von f .

„fv-Weg“ steht für „flussvergrößernder Weg“. Dabei handelt es sich um Wege im sogenannten „Restnetzwerk“.

Das Restnetzwerk G_f zu einem Netzwerk $G = (V, E, c)$ und einem Fluss f ist wie folgt definiert.

- O.B.d.A. nehmen wir an, dass das Netzwerk G keine entgegengesetzten Kanten hat, d.h.

$$(u, v) \in E \Rightarrow (v, u) \notin E .$$

- Für alle Paare $(u, v) \in V^2$ setze

$$rest_f(u, v) = \begin{cases} c(u, v) - f(u, v) & (u, v) \in E \\ f(v, u) & (v, u) \in E \\ 0 & \text{sonst} \end{cases}$$

- G_f hat die Knotenmenge V und die Kantenmenge

$$E_f = \{(u, v) \in V^2 \mid rest_f(u, v) > 0\} .$$

Ein fv-Weg ist ein einfacher Weg von q nach s im Restnetzwerk G_f . (einfacher Weg = Weg auf dem jede Kante höchstens einmal vorkommt)

Max-Flow = Min-Cut

Die Korrektheit der Ford-Fulkerson-Methode beruht auf dem „Max-Flow=Min-Cut“-Theorem.

Ein Cut oder Schnitt (Q, S) in einem Flussnetzwerk G ist eine Partitionierung der Knotenmenge $V = Q \dot{\cup} S$ mit $q \in Q$, $s \in S$.

- Die Kapazität des Schnitts (Q, S) ist definiert durch

$$c(Q, S) = \sum_{\substack{(u, v) \in E \\ u \in Q, v \in S}} c(u, v) .$$

- Der Fluss über den Schnitt (Q, S) ist definiert durch

$$f(Q, S) = \sum_{\substack{(u, v) \in E \\ u \in Q, v \in S}} f(u, v) - \sum_{\substack{(v, u) \in E \\ v \in S, u \in Q}} f(v, u) .$$

Lemma 2 [Max-Flow \leq Min-Cut]

Für jeden Fluss f und jeden Schnitt (Q, S) gilt $w(f) = f(Q, S) \leq c(Q, S)$. □

Dieses Lemma folgt direkt aus der Flusserhaltung. Formal zeigt man $w(f) = f(Q, S)$ per Induktion über die Größe der Menge Q (vgl. Übungen).

Satz 3 (Max-Flow=Min-Cut)

Die folgenden Aussagen sind äquivalent.

- a) f ist ein maximaler Fluss.
- b) Das Restnetzwerk G_f enthält keinen fv-Weg.
- c) Es gibt einen Schnitt (Q, S) mit $w(f) = c(Q, S)$.

Beweis von Satz 3:**aus a) folgt b):**

- Zum Zwecke des Widerspruchs, sei f ein maximaler Fluss und W ein fv-Weg in G_f .
- Dann kann f entlang von W vergrößert werden. Dies ist ein Widerspruch zur Maximalität von f .

aus c) folgt a):

Aus Lemma 2 folgt

$$w(f) = c(Q, S) \geq \text{Min-Cut} \geq \text{Max-Flow} ,$$

und somit gilt $w(f) = \text{Max-Flow}$.

aus b) folgt c):

- Aussage b) sagt, dass G_f keinen fv-Weg hat
- Definiere $Q = \{v \in V \mid \exists \text{ Weg von } q \text{ nach } v \text{ in } G_f\}$ und $S = V \setminus Q$.
- Da G_f keinen fv-Weg hat, folgt $s \in S$. Also ist (Q, S) ein wohldefinierter Schnitt mit $q \in Q$ und $s \in S$.
- Für jede Kante $(u, v) \in E$ mit $u \in Q$ und $v \in S$ gilt $f(u, v) = c(u, v)$, weil sonst $rest_f(u, v) > 0$ und somit $v \in Q$ wäre.
- Für jede Kante $(v, u) \in E$ mit $v \in S$ und $u \in Q$ gilt $f(v, u) = 0$, weil sonst $rest_f(u, v) > 0$ und somit wiederum $v \in Q$ wäre.
- Wir erinnern uns an die Definitionen von $c(Q, S)$ und $f(Q, S)$ und erhalten Aussage c), weil

$$\begin{aligned} c(Q, S) &= \sum_{\substack{(u,v) \in E \\ u \in Q, v \in S}} c(u, v) \\ &= \sum_{\substack{(u,v) \in E \\ u \in Q, v \in S}} f(u, v) - \sum_{\substack{(v,u) \in E \\ v \in S, u \in Q}} f(v, u) \\ &= f(Q, S) \stackrel{\text{Lemma 2}}{=} w(f) . \end{aligned}$$

□ Satz 3

Die Ford-Fulkerson-Methode terminiert, sobald kein fv-Weg mehr vorhanden ist. Aus Satz 3 a) und b) folgt somit, dass der berechnete Fluss f maximal ist.

Satz 4 (Korrektheit) Die Ford-Fulkerson-Methode berechnet einen maximalen Fluss.

Wir haben angenommen, dass die Kantenkapazitäten natürliche Zahlen sind. Unter dieser Annahme ist auch der durch die Ford-Fulkerson-Methode berechnete Fluss ganzzahlig, d.h. auf jeder Kante wird der Wert des Flusses durch eine natürliche Zahl beschrieben.

Korollar 5 (Ganzzahligkeit) Für jeden maximalen Fluss gibt es einen ganzzahligen Fluss mit gleichem Wert.

Diese Eigenschaft wird von vielen Anwendungen ausgenutzt (vgl. Übungen).

Laufzeit der Ford-Fulkerson-Methode

fv-Wege können beispielsweise durch Tiefen- oder Breiten-suche im Restnetzwerk gefunden werden. Damit dauert eine Iteration der Ford-Fulkerson-Methode nur Zeit $O(m)$. Sei $C = \sum_{e \in E} c(e)$. Da der Fluss in jeder Iteration um mindestens eine Einheit erhöht wird und C eine obere Schranke für den Wert des maximalen Flusses ist, können wir die Laufzeit der Ford-Fulkerson-Methode durch $O(mC)$ abschätzen und erhalten damit eine *pseudopolynomielle* Laufzeitschranke.

Im Allgemeinen hat die Ford-Fulkerson-Methode keine polynomielle Laufzeit. Im Folgenden werden wir jedoch sehen, dass die Berechnung der fv-Wege mittels einer Breitensuche eine polynomielle Laufzeit garantiert. Eine Breitensuche findet kürzeste fv-Wege, d.h. solche Wege zwischen q und s im Restnetzwerk, die eine minimale Anzahl Kanten enthalten.

Satz 6 (Edmonds und Karp, 1969) Die Laufzeit der Ford-Fulkerson-Methode mit Breitensuche ist $O(m^2n) = O(n^5)$.

Zum Beweis dieses Satzes müssen wir zeigen, dass der Algorithmus nach $O(mn)$ Iterationen terminiert.

Lemma 7 Von Iteration zu Iteration verringert sich die Distanz von q zu einem Knoten $v \in V$ im Restnetzwerk nicht.

Beweis: Entlang des fv-Weges können durch die Erhöhung des Flusses Kanten verschwinden und entstehen. Es gibt zwei Arten von Veränderungen.

- Für jede Kante (u, v) auf dem fv-Weg verringert sich $rest_f(u, v)$. Kanten mit minimaler Restkapazität auf dem fv-Weg, sogenannte Flaschenhalskanten, verschwinden aus dem Restnetzwerk, denn ihre Restkapazität wird auf 0 gesetzt.
- Gleichzeitig erhöht sich für jede Kante (u, v) auf dem fv-Weg die entgegengesetzte Restkapazität $rest_f(v, u)$. Falls $rest_f(v, u)$ zuvor den Wert 0 hatte entsteht eine neue Kante im Restnetzwerk.

Zum Zwecke des Beweises, verändern wir das Restnetzwerk in zwei Schritten und fügen zunächst nacheinander alle neuen Kanten hinzu und entfernen erst dann die Flaschenhalskanten.

Hinzufügen von neuen Kanten:

Kann das Hinzufügen der neuen Kanten Distanzen von der Quelle zu anderen Knoten verringern? – Nein!

Begründung: Eine Kante (v, u) wird nur dann eingefügt, wenn die Kante in umgekehrter Richtung – also die Kante (u, v) – auf einem flussvergrößerndem Weg liegt. Flussvergrößernde Wege sind aber kürzester Weg im Restnetzwerk. Wenn u also die Distanz ℓ von der Quelle hat, so hat v die Distanz $\ell + 1$ von der Quelle. Die neue Kante verbindet also einen Knoten mit Distanz $\ell + 1$ von der Quelle mit einem Knoten mit Distanz ℓ . Um aber die Distanz von der Quelle zu irgendeinem Knoten zu verringern, müsste man eine gerichtete Kante von einem Knoten mit Distanz k , für irgendein $k \geq 0$, zu einem Knoten mit Distanz $k + i$, für irgendein $i > 1$, einfügen.

Entfernen von Flaschenhalskanten:

Offensichtlich können Kantenlöschungen keine Distanzen verringern.

□ Lemma 7

Beweis von Satz 6:

- Wenn immer eine Kante (u, v) zur Flaschenhalskante wird, verschwindet sie aus dem Restnetzwerk. Sei ℓ die Distanz von q zu u zu diesem Zeitpunkt. Die Distanz von q zu v ist somit $\ell + 1$.
- Die Kante (u, v) kann in einer späteren Iteration wieder in das Restnetzwerk eingefügt werden und zwar wenn der Fluss auf Kante (v, u) erhöht wird. Dazu muss (v, u) auf einem kürzesten Weg liegen. Da die Distanz von q zu v sich aber nicht verringert hat, muss die Distanz von q zu u dann aber mindestens $\ell + 2$ sein.
- Entfernen und Wiedereinfügen einer Kante (u, v) erhöht die Entfernung von der Quelle zu u also um 2. Da die maximale Distanz $n - 1$ ist, kann eine Kante also nicht öfter als $\frac{1}{2}n$ mal entfernt werden.
- In jeder Iteration wird mindestens eine Kante entfernt. Es gibt bis zu $2m$ Kanten im Restnetzwerk. Also ist die Anzahl der Iterationen höchstens $\frac{1}{2}n \cdot 2m = nm$.

□ Satz 6

Der Algorithmus von Dinic, 1970

Idee: In jeder Iteration erhöhe den Fluss entlang von mehreren kürzesten fv-Wegen.

Dazu berechne aus dem Restnetzwerk $G_f = (V, E_f)$ ein Niveaunetzwerk $G'_f = (V, E'_f)$. Für $i \in \mathbb{N}_0$ sei

$$V_i = \{v \in V \mid \text{die Distanz von } q \text{ nach } v \text{ in } G_f \text{ ist } i\} .$$

G'_f enthält nur die Kanten von Niveau i zu Niveau $i + 1$, d.h.,

$$E'_f = \{(u, v) \in E_f \mid \exists i : u \in V_i, v \in V_{i+1}\} .$$

G'_f kann aus G_f durch Breitensuche in Zeit $O(m)$ berechnet werden.

Wir können G'_f als Flussnetzwerk auffassen, dessen Kantenkapazitäten den Restkapazitäten im Restnetzwerk G_f entsprechen.

- Sei ϕ ein Fluss im Niveaunetzwerk G'_f .
- Eine Kante $e \in E'_f$ heisst saturiert, wenn $\phi(e) = \text{rest}_f(e)$.
- ϕ heisst Sperrfluss wenn jeder q - s -Weg in G'_f mindestens eine saturierte Kante enthält.

Algorithmus zur Sperrflussberechnung:

- 1 $\forall e \in E'_f : \phi(e) = 0$;
- 2 Solange es einen q - s Weg W in G'_f gibt
- 3 erhöhe ϕ entlang von W soweit wie möglich;
- 4 entferne alle saturierten Kanten aus G'_f ;
- 5 Ausgabe von ϕ .

Lemma 8 Bei geeigneter Implementierung hat der Algorithmus zur Sperrflussberechnung eine Laufzeit von $O(nm)$.

Beweis:

Die Anzahl der in Schritt 2 berechneten q - s Wege ist höchstens m , da für jeden gefundenen Weg mindestens eine Kante aus G'_f entfernt wird. Die Berechnung eines Weges in Schritt 2 realisieren wir in Form einer Tiefensuche. Eine einzelne Tiefensuche kann jedoch bis zu $\Theta(m)$ Schritte benötigen. Auf diese Art und Weise würden wir nur eine Laufzeitschranke von $O(m^2)$ erhalten. Um die Laufzeitschranke $O(nm)$ einzuhalten, benötigen wir noch einen weiteren Trick. Wenn wir bei der Suche nach einem q - s -Weg in eine Sackgasse (= Weg der nicht zur Senke führt) laufen, so entfernen wir auch alle Kanten in dieser Sackgasse aus dem Niveaunetzwerk G'_f .

Amortisierte Laufzeitanalyse:

- Jede der m Kante wird höchstens einmal zur Sackgassenkante. Das verursacht Laufzeitkosten $O(1)$ pro Kante.
- Unter Vernachlässigung der Kosten für die vergebliche Exploration von Sackgassen, verbleiben für jeden der maximal m gefundenen q - s Weg nur Kosten $O(n)$, da jeder q - s Weg höchstens $n - 1$ Kanten enthält.

Gesamtkosten: $m \cdot O(1) + m \cdot O(n) = O(nm)$. □

Dinics Maximaler-Fluss-Algorithmus

- 1 $\forall e \in E : f(e) = 0;$
- 2 Solange es einen q - s Weg im Restnetzwerk gibt
- 3 berechne das Niveaunetzwerk $G'_f;$
- 4 berechne einen Sperrfluss ϕ in $G'_f;$
- 5 „addiere“ ϕ zu $f;$
- 6 Ausgabe von $f.$

- Beim Addieren der Flüsse in Schritt 5 ist zu beachten, dass Flüsse auf entgegengesetzten Kanten subtrahiert werden müssen.
- Die Korrektheit des Algorithmus folgt mittels des „Min-Cut=Max-Flow“-Theorems analog zur Ford-Fulkerson-Methode.

Satz 9 Die Laufzeit von Dinics Algorithmus ist $O(n^2m) = O(n^4).$

Beweis: Wir müssen zeigen, dass der Algorithmus nach $O(n)$ Iterationen terminiert.

Behauptung: Die Länge des kürzesten Weges im Restnetzwerk wächst von Iteration zu Iteration um mindestens eine Kante.

- Sei ℓ die Entfernung zwischen q und s zu Beginn der Iteration.
- Alle ursprünglichen Wege der Länge ℓ werden durch den Sperrfluss zerstört. Aber es können neue Wege durch neue erzeugte Kanten entstehen.
- Neue Kanten laufen jedoch entgegengesetzt zum Sperrfluss, also führen sie von Niveau i zu Niveau $i - 1$ für irgendein $i \geq 1$. Wege über solche Kanten haben Länge mindestens $\ell + 2$.

Somit gilt die Behauptung und der Satz folgt, weil die Länge eines q - s -Weges höchstens $n - 1$ ist. \square

Forward-Backward-Propagation (Malhotra, Kumar, Maheshwari, 1978)

Gegeben ein Niveaunetzwerk G'_f mit Restkapazitäten $rest_f(e)$, berechnen wir einen Sperrfluss ϕ in Zeit $O(n^2)$ statt $O(nm)$.

Für einen Knoten $v \in V$ bezeichne

- $A(v) = \{(v, u) \in E'_f\}$ die von v ausgehenden Kanten,
- $E(v) = \{(u, v) \in E'_f\}$ die in v eingehenden Kanten.

Wir starten mit $\phi = 0$ und erhöhen den Fluss ϕ nach und nach durch sogenannte „Forward-Backward-Propagationen“ bis wir einen Sperrfluss im Niveaunetzwerk berechnet haben.

Eine Propagationsphase besteht aus einer „Forward-“ und einer „Backward-Propagation“.

- Sei ϕ der berechnete Fluss zu Beginn einer solchen Propagationsphase.
- Das Potential einer Kante e ist definiert als

$$pot(e) = rest_f(e) - \phi(e) ,$$

entspricht also der noch ungenutzten Restkapazität der Kante e im Niveaunetzwerk.

- Das Potential eines Knotens v ist definiert als

$$pot(v) = \min \left\{ \sum_{e \in E(v)} pot(e), \sum_{e \in A(v)} pot(e) \right\} .$$

Forward-Propagation:

- Sei v^* der Knoten mit kleinstem, positivem Potential. Sei i das Niveau von v^* . Wir erzeugen $pot(v^*)$ Einheiten zusätzlichen Flusses am Knoten v^* .
- Dadurch erhalten wir einen Überschuss von $pot(v^*)$ Flusseinheiten am Knoten v^* . Diesen Überschuss schieben wir vorwärts entlang der Kanten in $A(v^*)$.
- Jetzt erhalten wir einen Überschuss auf einigen der Knoten auf Niveau $i + 1$. Diesen Überschuss verschieben wir wiederum zu Knoten auf Niveau $i + 2$, usw. bis wir die Senke erreichen.
- Da v^* der Knoten mit kleinstem Potential war, ist die Entsorgung des Überschussflusses sichergestellt, weil jeder Knoten genügend Potential hat, um den Überfluss weiterzuleiten.

Backward Propagation: analog – ausgehend vom selben Knoten v^* wird der Überfluss zur Quelle propagiert.

Forward-Propagation im Detail:

Wir benutzen eine FIFO-Queue Q zur Verwaltung der Knoten mit positivem Überschuss. Die Menge $A(v)$ der von v ausgehenden Kanten im Niveaunetzwerk wird als Liste verwaltet. $next(A(v))$ bezeichnet jeweils die nächste bisher noch nicht betrachtete Kante. Ziel ist die Berechnung eines Flusses ψ mit Wert $pot(v^*)$ vom Knoten v^* zur Senke s .

Algorithmus Forward-Propagation(v^*):

```

01  setze  $U(v^*) = pot(v^*)$ ; /* initialer Überfluss */
02  für alle  $v \neq v^*$  setze  $U(v) = 0$ ;
03  füge  $v^*$  in  $Q$  ein;
04  while  $Q \neq \emptyset$  do
05      nimm Element aus  $Q$  und nenne es  $v$ ;
06      while  $U(v) > 0$  do
07          sei  $e = (v, u) = next(A(v))$ ;
08          setze  $\psi(e) = \min\{pot(e), U(v)\}$ ;
09          setze  $U(v) = U(v) - \psi(e)$ ;
10          setze  $U(u) = U(u) + \psi(e)$ ;
11          falls  $u \neq s$  und  $u \notin Q$  füge  $u$  in  $Q$  ein;
12  Ausgabe von  $\psi$ .
```

Der Fluss ψ wird dann zum Fluss ϕ addiert.

Die Propagationsphasen werden solange wiederholt bis ein Sperrfluss berechnet ist. Die Kanten- und Knotenpotentiale werden dabei nach jeder Propagationsphase angepasst. Knoten und Kanten mit Potential 0 werden als saturiert bezeichnet und werden in den kommenden Propagationsphasen nicht mehr betrachtet, also aus dem Niveaunetzwerk entfernt. Die Sperrflussberechnung terminiert, sobald alle Knoten entfernt sind.

Beobachtung 10 Nach spätestens $n - 1$ Propagationsphasen ist ein Sperrfluss berechnet, weil in jeder Phase mindestens ein Knoten saturiert wird.

Lemma 11 Eine Forward-Propagation kann in Zeit $O(n + \ell)$ durchgeführt werden, wobei ℓ die Anzahl der neu saturierten Kanten ist. (Backward-Propagation analog.)

Beweis: Die Verwendung der FIFO-Queue Q garantiert die folgenden Eigenschaften.

- 1) Die Niveaus werden strikt nacheinander abgearbeitet.
- 2) Dadurch wird jeder Knoten höchstens einmal aus der Queue Q entnommen.
- 3) Für jeden entnommenen Knoten werden alle bis auf maximal eine der betrachteten Kanten saturiert.

Pro Knoten v gibt es somit maximal eine Kante in $A(v)$, die betrachtet aber nicht saturiert wird.

Die *Laufzeitkosten* werden auf die Kanten umgelegt:

- Für jede betrachtete Kante entstehen Kosten $O(1)$.
- Die Anzahl betrachteter, saturierter Kanten ist ℓ .
- Die Anzahl betrachteter, nicht saturierter Kanten ist höchstens n .

Damit sind die Gesamtkosten $O(n + \ell)$. □

Satz 12 Durch Forward-Backward-Propagation kann man einen Sperrfluss in Zeit $O(m + n^2) = O(n^2)$ berechnen. Dadurch erhält man einen Algorithmus zur Berechnung eines maximalen Flusses mit Laufzeit $O(n^3)$.

Beweis von Satz 12

- Beobachtung 10 liefert das nur höchstens $n - 1$ Propagationsphasen zur Sperrflussberechnung nötig sind. Sei ℓ_i die Anzahl der in der i -ten Propagationsphase gelöschten Kanten, $1 \leq i \leq n - 1$.
- Dann ergibt sich aus Lemma 11 die folgende Laufzeitabschätzung für die Sperrflussberechnung:

$$\sum_{i=1}^{n-1} O(n + \ell_i) = O\left(n^2 + \sum_{i=1}^{n-1} \ell_i\right) = O(n^2 + m) .$$

- Da $n - 1$ Sperrflussberechnungen einen maximalen Fluss liefern, folgt der Satz.

□